Apple IIGS
#74: Top Ten List Manager Things

Revised by: Dave Lyons May 1992
Written by: Jim Mensch November 1989

This Technical Note presents a method for speeding up custom List Draw routines, with sample source code for the APW assembler.
Changes since November 1989: Added information on `memFlag` and on shared `rListRef` resources, and noted that System 6.0 already checks the clip region and calls your `listDraw` routine only when needed.

Ten—More `memFlag` Bits

In each member record, bits 0 and 1 of `memFlag` indicate whether `memPtr` is a pointer, handle, or resource ID. You don't normally have to worry about that—a custom `listDraw` routine is one place that you do. The complete definition of `memFlag` is as follows:

Bit	Description
7	`memSelected`
6	`memDisabled`
5	`memNever` (Inactive)
4-2	reserved—set to zero
1-0	00 = `memPtr` is a pointer
	01 = `memPtr` is a handle
	10 = `memPtr` is a resource ID (type is `rPString` or `rCString`)
	11 = reserved

Nine—Sharing `rListRef` resources

When `listRef` is a resource ID, the List Manager calls `LoadResource` every time it needs your `rListRef` resource. If two or more lists share the same `rListRef`, they will get the same handle from `LoadResource` and will interfere with each other.

To give each list its own copy of your the `rListRef` resource, load the resource yourself and use `DetachResource`. Then feed the `listRef` to the List Manager as a handle. Repeat the process for each list.

Eight—Custom `listDraw` Routines and the Clip Region

The custom listDraw routine below speeds up your list when running System Software earlier than 6.0. The System 6.0 List Manager already calls your `listDraw` routine only for members that will not be completely clipped (but this is still a good starting point if you're writing a custom `listDraw` routine for some other reason).

To scroll text, the List Manager calls `ScrollRect` to scroll the list—then 6.0 redraws the newly-exposed members, and older versions redraw all the visible members. On small lists this is fine, but on larger lists it can cause the redrawing of much data that is already on the screen, which can take time. If your application does not require 6.0, you may want to use a custom `listDraw` routine like this one.

First, we check the current `clipRgn` (which the List Manager was kind enough to shrink down to include only the portion of the list that needs redrawing) against the passed item rectangle. If the rectangle is in any way enclosed in the `clipRgn`, then the member is redrawn; otherwise the routine simply returns to the List Manager without drawing. This sample routine is designed to work only with Pascal-style strings, but it can be easily modified to use any other type of string you choose.

```
MyListDraw   Start
;
; This routine draws a list member if any part of the member's
; rectangle is inside the current clipRgn.
;
; Note that the Data Bank register is not defined on entry
; to this routine.  If you use any absolute addressing, you
; must set B yourself and restore its value before exiting.
;
top          equ 0
left         equ top+2
bottom       equ left+2
right        equ bottom+2
rgnBounds    equ 2
;
oldDPage     equ 1
theRTL       equ oldDPage+2
listHand     equ theRTL+3
memPtr       equ listHand+4
theRect      equ memPtr+4
             using globals

             phd
             tsc
             tcd

             pha
             pha
             _GetClipHandle
             PullLong listHand

             ldy #2
             lda [listhand],y
             tax
             lda [listhand]
             sta listhand
             stx listhand+2

             lda [therect]              ; now test the top
             dec a                      ; adjust and give a little slack
             ldy #rgnbounds+bottom
             cmp [listhand],y           ; rgnRectBottom>=top?
```

```
            blt skip2
            brl NoDraw                  ; if not don't draw..
Skip2       ldy #bottom                 ; now see if the bottom is higher than the top
            inc a                       ; give a little slack
            lda [therect],y
            ldy #rgnBounds+top
            cmp [listhand],y
            blt NoDraw
NoTest      ANOP

            PushLong theRect
            _EraseRect            ; erase the old rectangle

            ldy #left
            lda [theRect],y
            tax
            ldy #bottom
            lda [theRect],y
            dec a
            phx
            pha
            _MoveTo
            ldy #2
            lda [memptr],y
            pha
            lda [memptr]
            pha
            _DrawString

            ldy #4
            lda [memPtr],y
            and #$00C0            ; strip to the 6 and 7 bits
            beq memDrawn          ; if they are both 0 the member is drawn
            cmp #$0080            ; member selected?
            bne noSelect          ; member not selectable
            PushLong theRect
            _InvertRect
            bra memDrawn
; if we get here the member is disabled
noSelect    PushLong #DimMask
            _SetPenMask
            PushLong theRect
            _EraseRect
            PushLong #NorMask
            _SetPenMask
memDrawn    ANOP


; exit here
            pld
            sep #$20
            longa off
            pla
            ply

            plx
            plx
            plx
            plx
            plx
            plx
            phy
            pha
            rep #$20
            longa on
            rtl
```

```
DimMask      dc  i1'$55,$AA,$55,$AA,$55,$AA,$55,$AA'
NorMask      dc  i1'$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF'
             end
```

Seven through One—Reserved For Future Expansion

Further Reference

- *Apple IIGS Toolbox Reference*, Volumes 1 and 3